НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено

Завідувач кафедри

_____ О.В. Коваль

(підпис) (ініціали, прізвище)

“___”_____2019р.

ДИПЛОМНА РОБОТА

на здобуття ступеня бакалавра

з напряму підготовки 6.050101 “Комп’ютерні науки “

на тему: Інтеграція динамічної бібліотеки математичних розрахунків з веб-сервісом Node.js

Виконав (-ла): студент (-ка) __4__курсу, групи___ТР-52___

Піддубняк Андрій Володимирович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник __доцент Демчишин Анатолій Анатолійович_____ _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Консультант _____ _____ _____
(назва розділу) (вчені ступінь та звання, прізвище, ініціали) (підпис)

Рецензент_____ _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2019

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший , бакалаврський

Напрям підготовки 6.050101 “ Комп’ютерні науки ”

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ О.В. Коваль
(підпис)
”___”_____2019р.

ЗАВДАННЯ
на дипломну роботу студенту

_____
(прізвище, ім’я, по батькові)

1. Тема роботи Інтеграція динамічної бібліотеки математичних розрахунків з веб-сервісом Node.js
керівник роботи_____доцент Демчишин Анатолій Анатолійович_____
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ”__”___ 201__р. № __

2. Строк подання студентом роботи _____

3. Вихідні дані до роботи персональний комп’ютер під керуванням операційної системи Microsoft Windows, мова програмування Javascript, веб-сервіс Node.js, мова програмування C++_____

4.Зміст розрахунково-пояснювальної записки (перелік завдань, які потрібно розробити)___проаналізувати існуючі програмні рішення та можливі засоби реалізації інтеграції, обґрунтувати обрані програмні застосунки та шляхи розробки програмних додатків, розробити програмне забезпечення, розробити користувацький інтерфейс, зробити висновки за результатами роботи_____

5. Перелік ілюстративного матеріалу
Інтеграція динамічної бібліотеки математичних розрахунків з веб-сервісом Node.js_____
_____
_____

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання ”___”_____201___р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Вивчення та аналіз задачі		
2	Розробка архітектури та загальної структури системи		
3.	Розробка структур окремих підсистем		
4.	Програмна реалізація системи		
5.	Оформлення пояснювальної записки		
6.	Захист програмного продукту		
7.	Передзахист		
8.	Захист		

Студент _____ _____
(підпис) (прізвище та ініціали,)

Керівник роботи _____ _____
(підпис) (прізвище та ініціали,)

АНОТАЦІЯ

Мета роботи — створення веб-сервісу для математичних розрахунків, реалізованих у вигляді динамічної бібліотеки на мові C++. Для цього досліджено технології створення бібліотеки та способи її інтеграції у Node.js середовище. Об’єктом роботи є інтеграція динамічної бібліотеки математичних розрахунків у веб-сервіс. Предметом роботи є створення веб-сервісу типу “клієнт-сервер”. Результати дослідження доповідалися на одній науковій конференції.

Записка містить 40 сторінок, 9 рисунків, один додаток і 36 посилань.

Ключові слова: FFI, NODE.JS, JAVASCRIPT, NODE-GYP, LU декомпозиція, КОМПІЛЯЦІЯ DLL БІБЛІОТЕКИ, ВЕБ-СЕРВІС.

ABSTRACT

The purpose of the work is to create a web-service for mathematical calculations implemented in the dynamic library in C ++. For this researcher, the technology of creating a library and how to integrate it into the Node.js environment. The object is the integration of a dynamic library of mathematical calculations in web services. The subject of the work is the creation of a web-service type "client-server". The results of the study were reported at one scientific conference.

The note contains 40 pages, 9 figures, one application and 36 links.

Key words: FFI, NODE.JS, JAVASCRIPT, NODE-GYP, LU decomposition, DLL COMPOSITION, WEB SERVICE.

ЗМІСТ

ВСТУП

На даний час технічного та інформаційного розвитку існує більше двадцяти популярних мов програмування [1] та технологій, що полегшують роботу з ними. Деякі з цих мов мають доволі схожу семантику, в той же час їх суттєва різниця полягає в можливостях, і, як наслідок, сферах застосування: наприклад, одні мови спеціалізуються для веб-розробки (JavaScript), інші (C#, C++) для створення десктоп додатків. У зв'язку з цим виникає потреба у раціональному використанні можливостей/переваг/швидкодії кількох мов в контексті одного проекту.

З плином часу все еволюціонує, як і мови програмування. Тому раніше були доволі популярні одні мови програмування, зараз їм на змінну прийшли нові. У зв'язку з цим, програмістам необхідно володіти актуальними мовами, технологіями, знаннями.

Як результат відбувається така ситуація, що програміст повинен писати один і той самий код, але за допомогою різних мов програмування. Добре коли це декілька стрічок, але коли потрібно переписувати доволі об'ємний алгоритм чи метод, виникає ряд проблем:

1. Копіювання коду може призвести до помилок, які будуть виявленні пізно і для виправлення буде затрачено багато ресурсів.
2. Переписування коду спеціалістом, який не писав попередній код може призвести до помилок.
3. Не раціональне використання людських ресурсів.
4. Не раціональне використання часу.

Особливо необхідно звернути увагу на останні два пункти, тому що в сучасних умовах необхідно раціонально використовувати людський ресурс та час. Адже можна використати ці ресурси більш ефективно.

C++ мова програмування була дуже популярна і нею володіла значна частина програмістів, зараз же прогрес не стоїть на місці, тому були створенні нові мови програмування, які краще використовувати для певних цілей. Для спеціаліста

необхідно опановувати їх для того, щоб працювати з актуальними мовами, технологіями та залишатись конкурентно-спроможними у своїй ніші.

З описаного вище, виникають ситуації, коли необхідно переписувати один і той самий код, але за допомогою різних мов.

Серед різноманіття клієнтських додатків важко знайти такі, що не використовують сторонні сервіси, бібліотеки, тощо. Тому для забезпечення додаткового функціоналу не завжди доречно використовувати його на клієнті. Оскільки немає необхідності виконувати ресурсозатратні операції на клієнті (а це може бути, як настільний комп'ютер, так і мобільний телефон), тому операції виконуються на віддаленому сервері, а доступ до них можливий завдяки запитам.

Мета роботи — створення веб-сервісу для математичних розрахунків, реалізованих у вигляді динамічної бібліотеки на мові C++. Для досягнення мети роботи було поставлено наступні завдання:

1. Проаналізувати існуючі способи інтеграції C++ коду у веб-середовище.
2. Розробити архітектуру взаємодії "клієнт – сервер".
3. Реалізувати веб-сервіс з інтегрованою бібліотекою математичних розрахунків.

Об'єктом роботи є інформаційні технології створення веб-сервісу типу "клієнт-сервер".

Предметом роботи є інтеграція динамічної бібліотеки математичних розрахунків у веб-сервіс.

Для реалізації серверу вибрано Node.js у звязку з тим, що за допомогою цієї технології легше реалізувати сервер, ніж за допомогою C++. Тобто, до нас вже були написанні модулі, які відповідають за підняття серверу та його стабільної роботи, на відміну від C++.

Для прикладу взято алгоритм LU декомпозиції, оскільки порядок величини займає $O(n^3)$, тому цей алгоритм є ресурсозатратним. Для інтерактивності проведено порівняння швидкостей виконання одного і того самого алгоритму, у вигляді нативного Javascript та скомпільованого C++ коду .

Код C++ компілюється у бібліотеку. Результатом роботи є сервіс, у який можна вставляти необхідний вам C++ код, компілювати його, та використовувати у Node.js середовищі, тобто на сервері.

Сервіс повністю налаштовано, тому його також можна буде використовувати для компіляції C++ коду у бібліотеку, для того, щоб ділитись з іншими програмістами. Як результат, компіляція відбувається лише на одній машині та результатом можна поділитись з іншими. Ось ще один спосіб використання моєї роботи.

Веб-сервіс Node.js може не тільки завантажувати бібліотеки Javascript, але й розширюватися за допомогою власних модулів (компілюються з C / C ++ [2] коду). Ці знання можуть стати в нагоді в конкретних випадках.

Використовуючи Node.js [3] можливо отримати прямий доступ до існуючих бібліотек C / C ++. Замість того, щоб викликати їх як зовнішні програми у стилі "виконати команду", можливо отримати код безпосередньо з існуючого вихідного коду і передати результати назад у Node.js у формі, зрозумілій для вашого середовища виконання Javascript. Таким чином можливий доступ до низькорівневого API операційної системи.

Є кілька практичних причин не переписувати код. Одна з них - відсутній код. Тобто, якщо при розробці використовувались застарілі інструменти для розробки проекту, початковий код цих інструментів часто втрачається. Похідною від цього є, коли застарілий код використовує залежності, які не можна переписати або змінити. Тому можна виділити декілька причин використання C++ скомпільованого коду у Node.js сервісі:

1. Необхідно реалізувати ресурсозатратний алгоритм, задачу тощо. Тому використання C++ буде доречним. Таким чином отримано доступ до низькорівневого API операційної системи, а як результат змога краще оптимізувати код, як мінімум завдяки строгій типізації та багатопоточністю C++. Як відомо, Node.js – однопоточний сервіс.
2. Наявний C++ код, який написаний раніше. Не завжди варто переписувати його, оскільки не можна бути впевненим на 100%, що задача може бути

досить складною в реалізації. Також це може бути одним з важливих складових/інструменту проекту, тому якщо переписувати C++ код – на проект накладається маса ризиків.

3. Сервіс Node.js досить ефективний, але якщо у проекті важливим фактором є швидкодія, тому не варто переписувати C++ код. Система повинна пропонувати готове та швидке рішення використання написаного раніше C++ коду, його компіляцію та подальшу інтеграції у Node.js середовище.

Отже використання C++ коду у Javascript середовищі є не таким нераціональним способом, як здається на перший погляд. Все-таки у цього методу є свої переваги.

Записка містить чотири розділи.

У першому описано декілька існуючих способів інтеграції c++ коду з Node.js сервісом.

У другому розділі дано технології імплементації сервісу математичних розрахунків.

У третьому розділі реалізовано інтеграцію динамічної бібліотеки математичних розрахунків з веб-сервісом Node.js.

У четвертому розділі описана методика роботи користувача з системою.

1. ОГЛЯД ІСНУЮЧИХ СПОСОБІВ ІНТЕГРАЦІЇ C++ КОДУ З NODE.JS СЕРВІСОМ

Є існуючий C++ код, який відіграє важливу роль у проекті, але необхідно отримати результат функції чи реалізувати складний алгоритм у веб-застосунку для реалізації внутрішнього процесу або публічно для будь-яких цілей. Проблема у тому, що код C++ не дуже легко використовувати у веб-застосунках.

Звичайно, можна писати частину веб-застосунків на C++ використовуючи CGI(це буде дуже довго).

CGI (Common Gateway Interface) [4] – стандарт інтерфейсу, який використовується для зв'язку програми з веб-сервером. Таку програму принято називати шлюзом (яка працює по такому інтерфейсу сумісно з веб-сервером). Хоча інколи називають таку програму – скрипт.

Хоча HTML (HyperText Markup Language) [5] статичний, веб-сторінка не може безпосередньо взаємодіяти з користувачем. До появи Javascript не було можливості відреагувати на дії користувачів, окрім як передавати введені дані на сервер для подальшого опрацювання. В випадку з CGI ця обробка виконується за допомогою зовнішніх програм та скриптів, доступ до яких виконується через стандартизований інтерфейс – загальний шлюз.

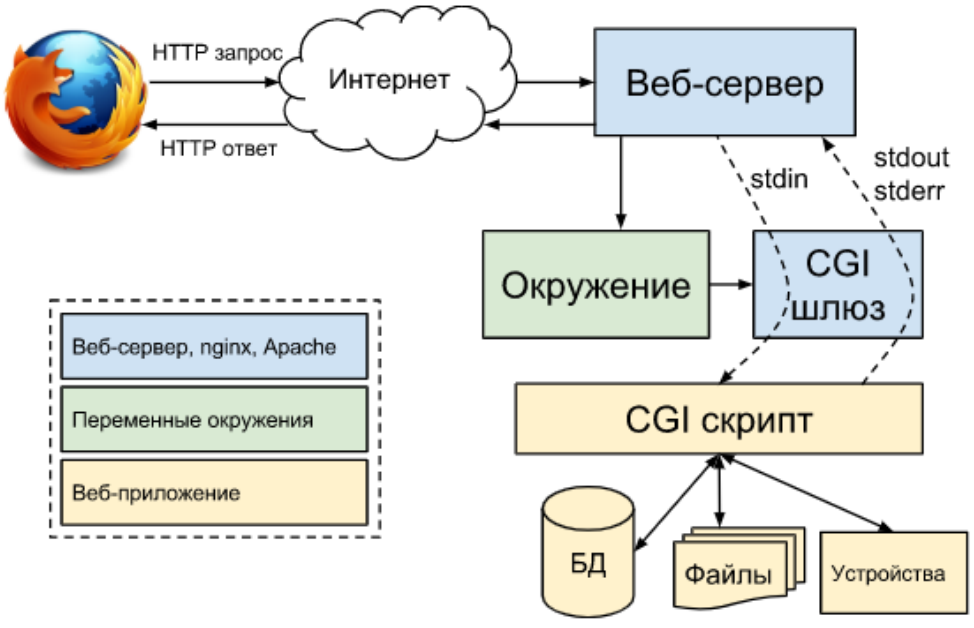

Рисунок 1.1, Ілюстрація роботи CGI

Власне інтерфейс реалізовано таким чином, що можна використовувати будь-яку мову програмування, яка може працювати з засобами вводу – виводу. Такими можливостями володіють навіть скрипти для вбудованих командних інтерпретаторів операційних систем, тому в простих випадках можна використовувати навіть командні скрипти.

Переваги CGI [6]:

1. Процес CGI скрипту не залежить від веб-серверу та у випадку помилки ніяк не відобразиться на роботі.
2. Може бути написаний на будь-якій мові програмування.
3. Підтримується більшістю веб-серверами.

Найбільшим недостатком технології є великі вимоги до продуктивності веб-серверу, оскільки кожне звернення до CGI-застосунку породжує новий процес зі всіма витікаючими звідси накладними витратами. Якщо ж воно написано з помилками, є шанс, що застосунок, наприклад, зациклиться. Браузер розірве з'єднання по закінченню тайм-ауту, але на серверній частині процес буде продовжуватись, поки адміністратор не зупинить його примусово.

Тому CGI не є доволі популярною технологію для роботи з веб-серверами, тому що в ньому бракує значних поліпшень продуктивності, що неприйнятно для веб-розробки на сьогодні. Якраз саме це і є важливим, що він вводить деякі проблеми з продуктивністю та масштабованістю проектів.

Також важливим фактором є те, що більшість веб-розробників не є програмістами C++, і якщо ультрависока продуктивність не є критичною для проекту, краще використовувати мови, що забезпечують більш високий рівень абстракції.

Є три загальні способи [3] інтеграції C++ коду із програмою Node.js (хоча в кожній категорії існує багато особливостей):

1. За допомогою консолі здійснювати виклик C ++ як окремий додаток у дочірньому процесі.

2. За допомогою бібліотеки упаковувати підпрограми C ++ у спільну бібліотеку (dll) і безпосередньо викликати ці процедури з Node.js.
3. За допомогою модулю Node.js є можливість компіляції коду C ++ як рідний модуль Node.js.

Кожен з цих варіантів має свої переваги і недоліки, вони в першу чергу відрізняються ступенем, в якому потрібно модифікувати C++, фактор продуктивності, який виникає при виклику скомпільованого коду, і комфорт у роботі з Node.js і API V8 [7].

Як вибрати? Найбільш очевидним питанням, яке потрібно задати, є доступ до джерела C++ або просто до двійкового файлу? Без вихідного коду потрібно сподіватися, що програма C ++ є або програмою командного рядка, або бібліотекою спільного доступу dll / lib.

1.1. Консоль

Якщо C ++ виконується окремо з командного рядка - або його можна зробити для цього, - можна запустити його за допомогою дочірнього Node API процесу. Ця опція працює для того, щоб принести майже все до Інтернету - це дійсно не має значення, якою мовою написано програму командного рядка, якщо просто запускається вона [8].

Дві особливості автоматизації роблять його привабливим. По-перше, оскільки виконується додаток C ++ в іншому процесі, по суті, виконується C ++ асинхронно - це велика перемога в Інтернеті, оскільки можна обробляти інший вхідний HTTP-трафік, поки програма C ++ працює. По-друге, дійсно не потрібно робити багато мовної інтеграції або використовувати складні V8 API.

1.2. Динамічна бібліотека

При автоматизації програми C ++, перевага в тому, що є дійсно чисте розділення між JavaScript та C ++. Автоматизація також дозволяє інтегруватися з майже кожною мовою програмування - до тих пір, поки вона може бути

автоматизована через stdin / stdout або вхідні та вихідні файли. Одним з недоліків є те, що C ++ дійсно є тільки одна точка входу. Безумовно, можна розробити складну координацію між C ++ та Node додатками, але автоматизація найкраще працює, коли просто необхідно відправити якийсь вхід до C ++ і чекати результатів [9].

Спільна бібліотека (або DLL) є відмінним рішенням у цій ситуації. Якщо C ++ вже в DLL, то можна почати відразу - але якщо ні, то необхідно скомпілювати код у DLL. Це досить легко - потрібно з'ясувати, які методи / функції будуть доступними для користувачів . Як тільки є DLL, використання інтерфейсу через Node.js є досить легким. Перетворення старих C або C ++ додатків у DLL може бути гарним інтеграційним вибором, коли автоматизація занадто громіздка. Він також дозволяє уникнути тонкощів розробки власних модулів C / C ++ для Node за допомогою API V8, який не завжди тривіальний.

1.3. Модуль

Якщо немає доступу до вихідного коду успадкованої програми C++, то консоль - найкращий варіант. Звичайно, якщо застарілий код не є C або C +++, то консоль також може бути кращим вибором (хоча є дійсно прив'язки від Node до інших мов). Якщо C / C ++ код вже знаходиться в dll або спільній бібліотеці, то, звичайно, це, мабуть, має найбільше сенсу використовувати FFI - як описано тут.

Для ситуацій, коли є повний доступ (і є зручним редагуванням), C/ C++, який необхідно реалізувати, створюючи рідний модуль [9], ймовірно, буде найпотужнішим підходом. По-перше, якщо код вже досить добре організований (чітко визначені точки входу та виходу / повернення), створити сам модуль не буде надто важко - особливо за допомогою Nan [10]. По-друге, модулі досить гнучкі - вони можуть бути блокуючими / синхронними або асинхронними, а також підтримувати більшість випадків використання (тобто передавати / повертати об'єкти, масиви і т.д.). Нарешті, коли створюється модуль Node.js, код JavaScript є чистішим, ніж під час використання автоматизації чи підходів спільної бібліотеки - порівнявши код JavaScript у цій публікації з іншими повідомленнями серії. В

кінцевому підсумку називаєте C ++, як якщо б він був повністю нормальним модулем JavaScript у Node.

Модулі Nan (Native addons for Node.js) [11] – динамічно пов’язані спільні об’єкти, реалізовані на мові C++ та також їх можна завантажити у Node.js середовище використовуючи функцію require(“”) [12]. Вони використовуються так само, як звичайні Node.js модулі (адже це і є Node.js модулем, але написаний вами). Модулі використовуються в основному для забезпечення інтерфейсу між Javascript, запущеним у бібліотеках Node.js та C/C++.

На сьогодні метод реалізації модулів є досить складним, оскільки розробнику необхідно володіти знаннями декількох різних компонентів, мов та API:

1. V8 LINK [13] : бібліотека C++. В даний час використовується для забезпечення реалізації Javascript. V8 надає можливість створювати об’єкти, функції, їх виклик, тощо. Перевагами V8 є: компіляція вихідного коду Javascript безпосередньо в власний машиний код, минаючи стадію проміжного байт-коду; Ефективна систему керування пам’ятю яка веде до швидкого виділення і маленьким паузам збірки сміття; зупинка виконання коду під час виконання збірки сміття; зменшує вплив призупинення застосунку при збірці сміття; Може точно оприділити, де в пам’яті знаходяться об’єкти, вказівники, тощо, що дозволяє уникнути витоку пам’яті при помилковій ідентифікації об’єктів у вигляді вказівників; введення прихованих класів і вбудованих кешей, що прискорюють доступ до властивостей і викликам функцій. V8 виконує Javascript сценарій в особливому контексті, який є окремою віртуальною машиною. Правда в одному процесі може працювати лише одна віртуальна машина, не дивлячись на можливість використання декількох потоків. В Chronium [14] (веб-браузер з відкритим вихідним кодом, на основі якого створюється ряд браузерів) це обходиться мультипроцесовою архітектурою, яка збільшує безпеку та стабільність, реалізуючи таким чином механізм пісочниці (спеціально виділене (ізольована) середовище для безпечного виконання комп'ютерних

програм. Зазвичай являє собою набір ресурсів для виконання гостьової програми - наприклад, місце на диску або в пам'яті, доступ до мережі, можливість спілкуватись з головною операційною системою, або зчитувати інформацію з пристроїв вводу-виводу або частково емулюються або дуже сильно обмежуються). Таким чином, не дивлячись на динамічну природу Javascript, розробникам вдалося застосувати методи, які характерні для реалізації класичних ООП мов програмування [15], такі як внутрішнє кешування, точний процес збору сміття, тощо. V8 відрізняється високою продуктивністю відносно інших бібліотек для роботи з Javascript.

2. Libuv [16] – бібліотека C++, яка реалізує цикл подій у сервісі Node.js, робочі потоки та асинхроні операції цієї платформи. Також служить для міжплатформної абстракції, що забезпечує легки доступ до POSIX (Portable Operating System Interface) [17] – набір інтерфейсів, що описують можливість роботи між операційною системою і прикладною програмною, бібліотеку мови C і набір додатків та інтерфейсів. Це стандарт, який був створений для сумісності різних UNIX ОС та транспортування прикладних програм на рівні байт-коду. Libuv також надає pthreads-подібну потокову абстакцію, яка може використовуватись для керування складними асинхроними додатками, які потребують виходу за межі стандартного циклу подій. Авторам цієї бібліотеки необхідно взяти до уваги, як уникнути блокування циклу подій за допомогою операції вводу-виводу або інших завдань шляхом завантаження роботи через libuv до неблокуючих системних операцій та робочих потоків.
3. Внутрішні бібліотеки Node.js є цими ж модулями, якці експортує C++ API, які і використовуються як модулі. Найважливішим серед них є клас ObjectWrap.

Сервіс Node.js включає в себе також безліч інших пов’язаних бібліотек. Ці бібліотеки знаходяться в каталозі deps, у кореневій папці Node.js. Тільки символи libuv, OpenSSL, V8 і zlib цілеспрямовано повторно експортуються у Node.js і можуть бути використані для різних цілей за допомогою модулів.

Висновки до розділу 1.

У цьому розділі проаналізовано способи інтеграції C++ коду з Node.js та обрано спосіб інтеграції за допомогою бібліотеки. Оскільки, після компіляції отримано бібліотеки, які можна використовувати у Windows та UNIX середовищі. Так як сервер може бути розташований у різних середовищах – був обраний варіант інтеграції за допомогою бібліотеки.

2. ТЕХНОЛОГІЇ ІМПЛЕМЕНТАЦІЇ СЕРВІСУ МАТЕМАТИЧНИХ РОЗРАХУНКІВ

Технологія Node.js була написана на C++, що є дуже швидким і потужним у розробці додатків для завдань низького рівня або важких обчислень.

Як результат, Node.js є дружнім для C++ і дозволяє розробникам запускати код C++ у цих програмах. Це робиться за допомогою модулів Node.js.

Модулі Node.js є динамічно пов'язаними спільними об'єктами, написаними на C++. Вони можуть бути завантажені в Node.js за допомогою функції require () і використовуватися так само, як якщо б вони були звичайним модулем Node.js. Вони використовуються в основному для забезпечення інтерфейсу між JavaScript, запущеним у бібліотеках Node.js і C++.

Модулі Node.js дають нам переваги у:

1. Можливість здійснювати інтенсивні, паралельні та високоточні розрахунки
2. Можливість використовувати бібліотеки C ++

2.1. Emscripten

Сервіс Emscripten [18] - компілятор LLVM-байткоду у JavaScript код, що може бути запущений у веб-браузері. Також він може бути отриманий з вихідного коду на мові C++.

Метою проекту Emscripten є створення та налагодження інструменту, що дозволив би виконувати в браузері код незалежно від мови програмування. Emscripten дозволяє здійснити компіляцію байткода LLVM в код на мові JavaScript, який може бути виконаний всередині веб-браузера, використовуючи тільки JavaScript-движок, без необхідності задіяння додаткових плагінів. Байткод LLVM може бути згенерований за вихідних текстів C++ за допомогою компіляторів llvm-gcc і clang, а також з коду на іншій мові програмування, для якого існує LLVM-

фронтенд. Emscripten надає змогу для трансляції бібліотеки SDL через canvas, а також підтримку OpenGL засобами WebGL.

Система LLVM (Low Level Virtual Machine) [19] - універсальна система аналізу, трансформації та оптимізації програм, що реалізує віртуальну машину з RISC-подібними інструкціями. Може використовуватися як оптимізуючий компілятор байткода в машинний код для різних архітектур або для його інтерпретації і JIT-компіляції (для деяких платформ).

В основі LLVM лежить проміжне представлення коду (Intermediate Representation, IR), над яким можна проводити трансформації під час компіляції, компонування і виконання. З цього уявлення генерується оптимізований машинний код для цілого ряду платформ, як статично, так і динамічно (JIT-компіляція). LLVM 3.6 підтримує статичну генерацію коду для x86, x86-64, ARM, PowerPC, SPARC, MIPS, Qualcomm Hexagon, NVPTX, SystemZ, Xcore. JIT-компіляція (генерація машинного коду під час виконання) підтримана для архітектур x86, x86_64, PowerPC, тощо.

Система LLVM написана за допомогою C ++ і перенесена на більшість Unix-подібних систем і Windows машин [20]. Модульна структура властива цій системі, окремі її модулі можуть бути вбудовані в різні програми, вона може розширюватися додатковими алгоритмами трансформації і кодогенератор для нових різних проектів

Окрім офіційних проектів LLVM, є дуже велика кількість інших проектів, які використовують засоби LLVM для різних завдань. За допомогою цих зовнішніх проектів можна використовувати LLVM для компіляції Ruby, Python, Java, D, PHP, Pure, Haskell, Lua, тощо. Основними перевагами LLVM є його універсальність, гнучкість і можливість повторного використання, саме тому використовується для такого широкого спектру різних завдань: від легких компіляцій JIT вбудованих мов, таких як Lua, до компіляції коду Fortran.

За допомогою Emscripten можна:

1. Перетворювати код на C і C ++ в код на JavaScript.
2. Перетворити в JavaScript код на будь-якому іншому мовою, який може бути трансльований в LLVM-байткод.

3. Перетворити середовища виконання інших мов, написані на C++, і запустити код, написаний на цих мовах (це вже робилося для Python і Lua)

Сервіс Emscripten дозволяє зробити нативний код доступним для використання в Web: платформа, що базується на стандартах, має незалежні сумісні реалізації і запускається всюди, з персональних комп'ютерів до iPad.

Використовуючи Emscripten, розробники C++ можуть уникнути портування коду вручну на JavaScript - і навіть уникнути вивчення JavaScript зовсім. Web-розробники теж виграють, так як вони можуть використовувати багато тисяч існуючих нативних утиліт і бібліотек на своїх сайтах.

Практично будь-який код на C і C ++ може бути скомпільовано в JavaScript с використанням Emscripten, починаючи з високопродуктивних ігор, які вимагають промальовування графіки, програють звуки і завантажують і обробляють файли, і закінчуючи фреймворками для створення додатків, наприклад, Qt.

Сервіс Emscripten генерує швидкий код, його формат за замовчуванням - asm.js, високооптимізуючи JavaScript, який в багатьох випадках може виконуватися зі швидкістю, близькою до нативної.

2.2. SWIG

Сервіс SWIG [21] (Simplified Wrapper and Interface Generator) - це засіб розробки програмного забезпечення для побудови інтерфейсів мови сценаріїв до програм C і C++. Спочатку розроблений у 1995 році, SWIG був вперше використаний вченими у відділі теоретичної фізики Національної лабораторії Лос-Аламосу для побудови інтерфейсів користувача до моделюючих кодів, що працюють на суперкомп'ютері Connection Machine 5. У цьому середовищі вченим необхідно було працювати з величезними обсягами даних моделювання, складним обладнанням і постійно мінливою кодовою базою. Використання інтерфейсу мови сценаріїв забезпечило просту, але дуже гнучку основу для вирішення цих проблем. SWIG спрощує розробку, значною мірою автоматизуючи завдання інтеграції мови

сценаріїв - дозволяючи розробникам і користувачам зосередитися на більш важливих проблемах.

Хоча SWIG була спочатку розроблена для наукових додатків, вона з тих пір перетворилася на інструмент загального призначення, який використовується в найрізноманітніших додатках - насправді майже все, що стосується програмування C / C ++.

Основна мета SWIG [22] - забезпечення можливості виконання функцій, написаних на одних мовах, з коду на інших мовах. Програміст створює файл .i з описом функцій, що експортуються; SWIG генерує вихідний код для склеювання C / C ++ і потрібної мови, створює виконуваний файл.

Вид виконуваного файлу залежить від вибраної мови:

1. виконуваний файл з вбудованим інтерпретатором скриптованої мови;
2. бібліотека, функції C / C ++ з якої автоматично стають доступні з іншої мови;
3. бібліотека функцій C / C ++ і бібліотека функцій - обгорток над функціями C / C ++ (наприклад, JNI для Java).

Двигуни скриптових мов вбудовують в програми на C / C ++ з наступних причин:

1. швидкість розробки за допомогою скриптової мови вище, ніж швидкість розробки на C / C ++;
2. користувачі отримують можливість автоматизації своїх дій за допомогою сценаріїв. Наприклад, в іграх сценарії використовуються для написання сюжету і рівнів;
3. розробники отримують можливість автоматизації для тестування ПО на етапі розробки. Готове ПО може не включати скриптова движок.

Причини створення бібліотек функцій C / C ++, доступних інтерпретаторів інших мов:

1. надання функціональності, відсутньої в скриптовій мовою;

2. оптимізація найбільш часто виконуваних ділянок коду для підвищення продуктивності.

Сервіс SWIG використовується різними способами:

1. Створення більш потужних C / C ++ програм. За допомогою SWIG можна замінити функцію main () програми C інтерпретатором сценаріїв, з якого можна керувати програмою. Це додає багато гнучкості і робить програму "програмованою". Тобто інтерфейс сценаріїв дозволяє користувачам і розробникам легко змінювати поведінку програми без необхідності змінювати код низького рівня C / C ++. Переваги цього численні. Насправді, подумайте про всі великі пакети програм, які використовується щодня - майже всі вони включають спеціальну мову макросу, мову конфігурації або навіть механізм сценаріїв, який дозволяє користувачам робити налаштування.
2. Швидке прототипування і налагодження. SWIG дозволяє C / C ++ програмам розміщуватися в середовищі сценаріїв, які можна використовувати для тестування та налагодження. Наприклад, можна протестувати бібліотеку з набором скриптів або використовувати інтерпретатор сценаріїв як інтерактивний відладчик. Оскільки SWIG не вимагає модифікації базового коду C / C ++, його можна використовувати, навіть якщо кінцевий продукт не покладається на сценарії.
3. Системна інтеграція. Мови сценаріїв працюють досить добре для управління і склеювання слабко пов'язаних компонентів програмного забезпечення. З SWIG, різні C / C ++ програми можуть бути перетворені в модулі розширення мови сценаріїв. Ці модулі можуть бути об'єднані разом для створення нових і цікавих додатків.

4. Побудова модулів розширення мови сценаріїв. SWIG може використовуватися для перетворення звичайних бібліотек C / C ++ на

компоненти для використання в популярних мовах сценаріїв. Звичайно, все одно краще переконатися, що ніхто ще не створив модуль перед цим.

2.3. Node-gyp

Інструмент Node-gyp [23] є крос-платформним інструментом командного рядка, написаним у Node.js для компіляції модулів для Node.js. Вона поєднує проект gyp, який використовується командою Chromium, і знімає біль, пов'язану з різними відмінностями в будівельних платформах. Це заміна програми node-waf, яка видаляється для вузла v0.8. Якщо у вас є рідний аддон для вузла, який все ще має файл wscript, то обов'язково потрібно додати файл binding.gyp для підтримки останніх версій вузла.

Підтримуються кілька цільових версій вузла (4, 5, 6 тощо), незалежно від того, яка версія вузла фактично встановлена у вашій системі (node-gyp завантажує необхідні файли розробки або заголовки для цільової версії).

Особливості:

1. Простий у використанні, послідовний інтерфейс
2. Ті самі команди для побудови модуля на кожній платформі
3. Підтримує кілька цільових версій Node.js

Вам також знадобляться ці три інструменти, встановлені на вашій машині (якщо ваше середовище розробки Windows, інакше на Unix-машинах ці програми встановлені за замовчуванням):

1. make
2. g ++
3. Python 2.7

Сервіс make [24] - утиліта, яка автоматизує процес перетворення файлів з однієї форми в іншу. Найчастіше це компіляція вихідного коду в об'єктні файли і подальша компонування у виконувані файли або бібліотеки. Утиліта використовує make-файли, в яких вказані залежності файлів один від одного і правила для їх

використання один з одним. На основі інформації про час останнього зміни кожного файлу make визначає і запускає необхідні програми. Найчастіше, makefile містить вказівки для утиліти make про те, як компілювати і компонувати програму. Наприклад, для мов програмування C / C ++ вихідний файл C або C ++ повинен бути перекомпілірovать при кожному своєму зміні. Якщо змінюється заголовки, кожен C / C ++ вихідний файл, що включає його, повинен бути перекомпілірovать, щоб бути безпечним. Кожна компіляція створює об'єктний файл, відповідний вихідного файлу. Нарешті, якщо вихідний файл був скомпільований, то все об'єктні файли, будь то новостворені або збережені з попередніх компіляцій, повинні бути скомпоновані разом, щоб зібрати новий виконуваний файл програми [1]. Ці інструкції разом зі своїми залежностями вказані в makefile. Якщо з останньої компіляції жоден з файлів, необхідних для складання, не змінювався, то при компіляції нічого не відбувається. Для великих проектів, використання make-файлів може значно скоротити час складання додатків при зміні всього лише кількох вихідних файлів.

Файли make складаються з 5 складових: явних правил, неявних правил, визначень змінних, директив та коментарів.

1. Явна правило повідомляє, коли і як потрібно перетворити один або кілька файлів, які називаються цільовими правилами. У явному правилі перераховуються й інші файли, від яких залежать цілі, звані залежностями мети, а також вказується спосіб, який використовується для створення або поновлення мети.
2. Негласне правило повідомляє, коли і як перетворити клас файли, виходячи з їх назви. Воно містить опис, як мета може впливати на файли з ім'ям, схожим на мету, і вказує спосіб для створення або поновлення такої мети.
3. Визначення змінної є рядком, в якій вказується текстове строкове значення змінної, яка може бути підставлена в текст пізніше.
4. Директива вказує утиліті make на виконання чого-небудь особливого при читанні файлу makefile (наприклад, читання іншого make-файлу).

5. Коментарі в рядку makefile починаються з символу '#'. Сам коментар і решта рядка ігнорується.

Утиліта G++ [25] - традиційне позначення GNU C++, вільно поширюваного компілятора мови C++. Є частиною GCC - колекції компіляторів GNU. На операційних системах Unix, команда gcc зазвичай використовується для виклику компілятора GCC, як і команда g++. За замовчуванням мову програмування C або C++ визначається по розширенню компілюючого. Але при цьому G ++ все одно є компілятором, а не просто препроцесором. G ++ будує об'єктний код безпосередньо з вихідного C ++ коду, без використання проміжної C-версії (на відміну від деяких компіляторів, які створюють об'єктний код Сі з початкових кодів, написаних на C ++). Це дозволяє отримувати більш повну і точну інформацію про роботу програми при налагодженні (наприклад, при використанні GDB).

Мова програмування Python [26] - високорівнева мова програмування загального призначення, орієнтована на підвищення продуктивності розробника і читання коду. Синтаксис ядра Python мінімалістичний. У той же час стандартна бібліотека включає великий обсяг корисних функцій. Оскільки Node-gyp написаний за допомогою python мови, для його компіляції нам необхідно встановити на машину компілятор Python 2.7.

Висновки до розділу 2.

У цьому розділі досліджено технології імплементації бібліотеки у Node.js середовище. Обрано Node-gyp, тому що для компіляції C++ коду необхідно найменше ресурсів.

3. РЕАЛІЗАЦІЯ ІНТЕГРАЦІЇ ДИНАМІЧНОЇ БІБЛІОТЕКИ

3.1. Архітектура сервісу

У цьому розділі показано, як створити сервіс, що компілю’ C++ динамічну бібліотеку для подальшого використання у Node.js сервісі. Метою є створення веб-сервісу типу “клієнт – сервер” (рис.3.1).

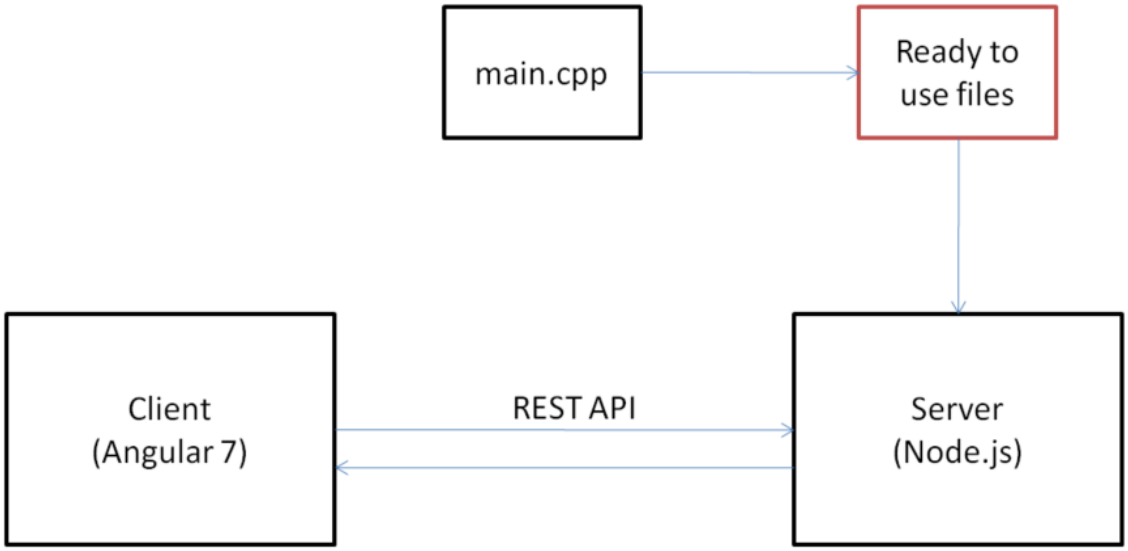

Рис. 3.1. Схема веб-сервісу

Клієнт реалізовано за допомогою фреймворку Angular 7 [27]. Angular 7 – фреймворк створений компанією Google для побудови клієнтських веб-застосунків. Він використовується для створення SPA (Single Page Application) рішень, тобто односторінкових застосунків. Фреймворк надає таку функціональність як двостороннє зв’язування, що надає змогу динамічно змінювати об’єкти в одному місці інтерфейсу при зміні данних моделі у іншому, шаблони, маршрутизація, тощо. Головною особливістю фреймворку є те, що в якості мови програмування він використовує Typescript мову програмування. Typescript – мова, яка є сумісною з Javascript та компілюється в нього. Після компіляції програми, її можна використовувати у будь-якому сучасному браузері або ж разом з веб-сервісом

Node.js. Головні відміності Typescript від Javascript полягають у тому, що перший має явне статичне назначення типів, підтримка повноціного використання класів, а також підключення модулів, що збільшує швидкість роботи, полегшує читаємість, повторне використання коду, рефакторинг.

Typescript [28] виник через явні недоліки Javascript, в великомасштабних застосунках. Проблеми з розробкою високонавантажених програм на JavaScript привели до необхідності полегшення розробки компонентів мови. Під час розробки TypeScript шукали таке рішення, яке не порушуватиме його крос-платформної підтримкою і сумісність зі стандартом. Знаючи, що тільки стандарт ECMAScript пропонує підтримку в майбутньому для програмування на базі класів. TypeScript був заснований на цьому припущенні, що призвело до створення компілятора JavaScript з набором синтаксичних мовних рішень, збільшених на основі пропозиції, яке реалізує розширення в JavaScript. У цьому сенсі TypeScript є готовим уявленням про те, що очікувати від ECMAScript 6. Унікальний аспект не в реченні, а в додаванні в TypeScript статичної типізації, що дозволяє статично аналізувати мову, полегшуючи розробку та підтримку.

Typescript – надлаштування над Javascript. Тобто програма Javascript також є корректною програмою Typescript та навпаки. У Typescript можна використовувати вже налагоджений Javascript, його модулі, бібліотеки, тощо. Компілятор Typescript – tsc. Запукається на будь-якому движку Javascript. Також може використовуватись у Node.js середовищі, як серверу, у вигляді пакету. Остання версія компілятору використовує ES5, але також надано можливість користуватись і ES6, що надає доступ до унікальних можливостей цієї версії, як наприклад, класи, генератори, проміси, тощо.

Веб-сервіс Node.js – середовище виконання Javascript коду, яка побудована на движку Javascript Chrome V8, що надає змогу перетворювати виклики на мові Javascript в машиний код. Node.js перш за все створено для побудови серверних застосунків за допомогою Javascript мови програмування. Хоча також можна розробляти десктоп додатки за допомогою фреймворку Electron [29], та навіть написання коду для мікроконтролерів. Іншими словами, Node.js надає можливість

писати високопродуктивний та легко масштабуємий код з використанням Javascript. Як зазначалося раніше, Node.js – однопоточка, але під капотом бібліотека libuv використовує засоби для роботи з багатопоточністю.

Для полегшення роботи з Node.js створено фреймворк Express.js. Він надає ряд готових рішень та абстракцій, які полегшують роботу створення серверу, серверної логіки, роботу з кукі, обробку даних, роботу з базами данних, політики cors, middlewares, тощо.

3.2. Підготовка бібліотеки

Перш за все необхідно створити бібліотеку. Для прикладу взято алгоритм LU декомпозиції [30]. LU декомпозиція – у чисельному аналізі та лінійній алгебрі, нижньому - верхньому (LU) коефіцієнтах розкладання чи факторизації - матриця як добуток нижньої трикутної матриці та верхньої трикутної матриці (рис. 3.1.). Продукт іноді включає матрицю перестановок. LU розкладання можна розглядати як матричну форму гауссової елімінації. Комп'ютери зазвичай вирішують квадратні системи лінійних рівнянь, використовуючи LU розкладання, і це також є ключовим кроком при інвертуванні матриці або обчислення детермінанта матриці.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}$$

Рис. 3.1. LU декомпозиція

Для вирішення системи лінійних рівнянь необхідно:

1. Отриманий LU-розклад матриці A (матриця коефіцієнтів) використовується для вирішення систем лінійних рівнянь з різними векторами b в правій частині:

$$Ax = b$$

2. A = LU, якщо відомо LU-розклад матриці A. Початкова система буде записана як:

$$LUx = b$$

3. Зазвичай такі система мають рішення у два кроки. Перший:

$$Ly = b$$

4. L – нижня трикутна матриця, тому система вирішується шляхом прямої підстановки. На другому кроці обраховується система:

$$Ux = y$$

Оскільки U – верхня трикутна матриця, то система обраховується шляхом зворотньої підстановки

Обернення матриці A еквівалентно розвязку лінійної системи: AX = I. Тобто X – невідома матриця, I – одинична матриця. Рішення X системи є оберненою матрицею A^{-1}. Систему можна вирішити описаним вище методом LU декомпозиції.

Реалізовано метод LU декомпозиції, оскільки він є ресурсоємким та займає $O(n^3)$ операцій. Саме через це обрано цей метод для того, щоб порівняти, чи втратить скомпільований C++ код у швидкодії. Очікується якщо не приріст швидкодії, то такий самий рівень. Приріст очікується за рахунок того, що код написано на низькорівневій мові, з строгою типізацією, багатопоточністю, тощо. І якщо необхідно реалізувати ресурсоємку операцію, то краще її виконувати за допомогою C++. Всерівно є точка в продуктивності, де зустрічаються C++ та Javascript, але C++ йде вгору, на відміну від Javascript. Як результат, алгоритм було досліджено, вивчено всі можливі варіанту вирішення алгоритму, розвитку подій, та реалізовано на C++, та за допомогою Node.js сервісу. Це було зроблено для того,

щоб потім можна було виконати ці дві функції, зробити заміри, та визначити, скільки часу затрачено на виконання цих функцій, яка з них відпрацювала швидше, яке співвідношення часу виконання C++ коду та Javascript.

3.3. Налаштування конфігураційного файлу

Конфігураційний файл [31] - інструмент для проекту Chromium, який генерує нативні Visual Studio, Xcode і SCons та / або створює файли збірки з незалежного від платформи вхідного формату. Формат введення необхідно створити якомога більш загальним, не витрачаючи додаткового часу на те, щоб «отримати все правильно», за винятком випадків, коли це не призвело б до появи помилок в кінцевому результаті.

Для коректної компіляції динамічної бібліотеки необхідно створити файл bindings.gyp (рис. 3.2.). Всередині нього створити JSON об’єкт, створити поле “sources” в якому вказати файл динамічної бібліотеки, який готовий для компіляції. Створити поле ““target_name” в якому вказуємо назву кінцевих файлів. Далі, якщо необхідно, створити поле “include_dirs”, в якому вказується, які залежності будуть використовуватись у скомпільованому коді. Також є безліч умов, які ще можна створити.

Файл описується за допомогою мови Python. (Це насправді JSON, з двома малими відхиленнями Pythonic: коментарі вводяться з #, а коми ставляться після останнього елемента у списку або словнику.

Поля верхнього рівня у файлі .gyp є такими:

“Variables”: Визначення змінних, які можна інтерполювати і використовувати в інших частинах файлу.

“includes”: список інших файлів, які будуть включені до цього файлу. За умовами, включені файли мають суфікс .gypi (gyp include).

“target_defaults”: Налаштування, які будуть застосовані до всіх цілей, визначених у цьому файлі .gyp.

“target”: список цілей, для яких цей файл .gyp може генерувати збірки. Кожна мета є словником, який містить параметри, що описують всю інформацію, необхідну для побудови цілі.

“conditions”: список специфікацій умов, які можуть змінювати вміст елементів у глобальному словнику, визначених цим файлом .gyp, на основі значень різних змін. Як випливає з наведеного вище прикладу, найбільш поширене використання розділу умов у словнику верхнього рівня полягає в тому, щоб додати до списку цілей специфічні для платформи цілі.

Переважна більшість цілей - бібліотеки. приклад бібліотеки, включаючи додаткові функції, які повинні охоплювати більшість потреб бібліотек:

```
{
  'targets': [
    {
      'target_name': 'foo',
      'type': '<(library)'
      'msvs_guid': '5ECEC9E5-8F23-47B6-93E0-C3B328B3BE65',
      'dependencies': [
        'xyzzy',
        '../bar/bar.gyp:bar',
      ],
      'defines': [
        'DEFINE_FOO',
        'DEFINE_A_VALUE=value',
      ],
      'include_dirs': [
        '..',
      ],
      'direct_dependent_settings': {
        'defines': [
          'DEFINE_FOO',
          'DEFINE_ADDITIONAL',
        ],
        'linkflags': [
        ],
      },
      'export_dependent_settings': [
        '../bar/bar.gyp:bar',
      ],
      'sources': [
        'file1.cc',
        'file2.cc',
      ],
      'conditions': [
        ['OS=="linux"', {
          'defines': [
            'LINUX_DEFINE',
          ],
          'include_dirs': [
            'include/linux',
          ],
        ],
        ['OS=="win"', {
          'defines': [
            'WINDOWS_SPECIFIC_DEFINE',
          ],
        }, { # OS != "win",
```

```
            'defines': [
              'NON_WINDOWS_DEFINE',
            ],
```

Рис. 3.2. Приклад конфігураційного файлу

Можливі записи в цілі бібліотеки в значній мірі такі ж, як і ті, що можуть бути вказані для виконуваного цілі (визначає, include_dirs і т.д.). Відмінності включають:

“type”: цей параметр майже завжди повинен бути встановлений на "<(library)", що дозволяє користувачеві визначати, в якому випадку бібліотеки повинні бути побудовані статично або спільно. (У Linux, принаймні, прив'язка до спільних бібліотек зберігає значне час зв'язку.) Якщо необхідно встановити тип бібліотеки, яку потрібно побудувати, тип може бути встановлений явно для static_library або shared_library.

“direct_dependent_settings”: Це визначає параметри, які будуть застосовані до інших цілей, які безпосередньо залежать від цієї мети - тобто, перелічують цю ціль у налаштуваннях 'залежностей'. Тут перераховуються визначення, include_dirs, cflags і linkflags, які інші цілі, які компілюють або посилаються на цю ціль, повинні будувати послідовно.

“export_dependent_settings”: У ньому перелічено цілі, чиї прямі_залежні_настройки повинні бути передані іншим цілям, які використовують (залежать) від цієї цілі.

3.4. Підготовка Node.js середовища

В цьому підрозділі описується підготовка Node.js середовища для використання скомпільованого коду. Динамічна бібліотека компілюється в директорію “build”, в цій директорії створюється ще одна директорія “Release”, саме тут і будуть знаходитись скомпільовані файли. Буде декілька файлів з однією назвою, але з різним розширенням. Це зроблено для того, щоб можна використовувати скомпільований код у різних середовищах (Windows, Unix) і залежно від середовища використовувати відповідні файли.

Для використання скомпільово декілька файлів, розширенням .dll та .os. З першим розширенням файли використовуються у Windows середовищі, а останні у UNIX.

Для коректного завантаження бібліотеки необхідно скористатися npm пакетом “node-ffi” [32]. “node-ffi” є модулем Node.js для завантаження та виклику динамічних бібліотек за допомогою чистого JavaScript (рис. 3.3.). Він може бути використаний для створення прив'язок до рідних бібліотек без написання будь-якого C ++ коду.

Вона також спрощує код node.js за допомогою коду C, оскільки вона займається перекладом типів через JavaScript і C, які можуть додавати коду шаблону коду. Необхідно розуміти, як працює цей пакет, тому що можна досить легко створювати ситуації, коли не будете розуміти що відбувається. Ось як відбувається завантаження бібліотеки:

```
Library = ffi.Library('./build/Releaseresult', {
        "LUDecomposition": [int, [floatArrayType, floatArrayType, floatArrayType,
int, int]]
})
```

Рис. 3.3. Приклад завантаження бібліотеки

Спочатку на рисунку 3.3 вказуємо шлях до директорії, де лежить скомпільована бібліотека. Далі необхідно вказати назву функції, яка буде використовуватись у майбутньому, також першим параметром вказуємо тип змінної, який функція повертає після свого виконання. Другим параметром передається масив змінних, які містять в собі інформацію, якого типу змінні необхідно передати, як аргументи, у функцію для коректної її роботи.

Для створення цих змінних необхідно використати npm пакет “ref” [33] (рис. 3.4.). Цей модуль створено на основі старого класу Pointer з node-ffi, але з метою використання швидких екземплярів Nup's Buffer замість повільного класу C ++ Pointer. Ці дві концепції раніше були дуже схожі, але тепер цей модуль переносить функціональні можливості, які показали покажчики, і буфери відсутні, так що тепер буфери є набагато більш потужними.

Особливості:

1. Отримати адресу пам'яті будь-якого екземпляра буфера
2. Читання / запис посилань на об'єкти JavaScript в екземпляри буфера
3. Адреси зчитування / запису екземплярів буфера звертаються до інших екземплярів буфера
4. Читання / запис значень даних int64_t і uint64_t (чисел або рядків)
5. "Типова" конвенція, так що можна вказати буфер як int *, а посилання / dereference за бажанням.
6. Пропонує буферний екземпляр, що представляє NULL покажчик

Таким чином за допомогою цього пакету створюються змінні певного типу, щоб у майбутньому вказати їх під час завантаження бібліотеки. Ось як це відбувається:

```
const {array} = req.body;
var ffi = require ("ffi")
var ref = require ("ref")
var float = ref.types.float;
var arrayType = require ("ref-array")
var floatArrayType = ArrayType (float);
```

Рис. 3.4. Імпорт модулів

Спочатку отримується з клієнта масив, з яким необхідно виконати алгоритм LU декомпозиції. Далі оголошуються npm пакети “ffi” та “ref”. Наступне –створення змінної float, що вона в собі буде зберігати тип float. Оскільки числа в матриці можуть бути нецілі, тому вибираємо саме цей тип. Далі створюється змінна “arrayType” і вказується, що в собі вона зберігає тип масиву. І останнє, створення змінної, в якій вказуємо, що вона має тип масиву, та всі елементи у ній будуть нецілі числа.

Після цього необхідно завантажити бібліотеку, як наведено вище.

Наступне, запуск функції з скомпільованої бібліотеки “LUDecomposition”, в результаті якої отримано масив з двох матриць (верхньої та нижньої). Також переваги використання такої бібліотеки полягають у тому, що вона повністю вписується у Node.js середовище. Тобто локальні чи глобальні змінні можна

передавати у функцію, та мати доступ до них по посиланню, тобто не потрібно їх передавати у функцію та повертати з функції.

Але перед тим як запустити виконання функції з бібліотеки, створюється змінна, в яку записується час в данний момент. Це використовується для того, щоб обчислити скільки часу займе виконання функції. Замір часу відбувається за допомогою Node.js модулю “performance”. Метою модулю є підтримка збору показників ефективності з високою роздільною здатністю. Це той самий API продуктивності, що й у сучасних веб-браузерах. Після виконання функції необхідно знову визначити який час в данний момент, та відняти з попереднім, що отримано перед виконанням функції.

Після цього виконуються аналогічні дії, але вже використовуємо функцію реалізовану за допомогою нативного Javascript. Також не забуваємо обчислити час, який займе виконання функції.

Після цього засобами REST API надсилається відповідь до клієнта.

Система REST API (Representational State Transfer) [34] - стиль взаємодії компонентів розподіленого додатка до мережі. REST являє собою узгоджений набір обмежень, що підтримують при проектуванні розподіленої системи. У визначених випадках (інтернет-магазинах, пошукових системах, системах даних, основані на даних) це призводить до підвищення рівня продуктивності та спрощення архітектури. У широкому сенсі, компоненти в REST взаємодіють зразок взаємодії клієнтів і серверів у Всесвітній павутині. REST є альтернативною RPC .

В мережі Інтернет виклик віддаленої процедури може являти собою звичайний HTTP-запит (зазвичай “GET” або “POST”; такий запит називають “REST-запит”), а необхідні дані передаються в якості параметрів запиту.

Для веб-служб, побудованих з урахуванням REST (тобто не порушують накладаються їм обмежень), застосовують термін “RESTful”.

На відміну від веб-сервісів (веб-служб) на основі SOAP, не існує “офіційного“ стандарту для RESTful веб-API. Справа в тому, що REST є архітектурним стилем, в той час як SOAP є протоколом. Незважаючи на те, що REST не є стандартом сам по

собі, більшість RESTful-реалізацій використовують стандарти, такі як HTTP, URL, JSON і XML.

Першим обмеженням, які можуть застосовуватися до гібридної моделі, є приведення архітектури до моделі клієнт-сервер. Розмежування потреб є принципом, що лежить в основі даного обмеження. Відділення потреби інтерфейсу клієнта від потреб сервера, що зберігає дані, підвищує переносимість коду клієнтського інтерфейсу на інші платформи, а спрощення серверної частини покращує масштабованість. Найбільше ж вплив на всесвітню павутину, мабуть, має саме розмежування, яке дозволяє окремим частинам розвиватися незалежно один від одного, підтримуючи потреби в розвитку інтернету з боку різних організацій.

Висновки до розділу 3.

У розділі досліджено проведення інтеграції бібліотеки у Node.js середовище, відображено налаштування Node.js середовища, та опис алгоритму LU декомпозиції, який реалізовано у динамічній бібліотеці.

4. МЕТОДИКА РОБОТИ КОРИСТУВАЧА З СЕРВІСОМ

Для прикладу було створено веб-сервіс типу “клієнт – сервер” (рис. 4.1.).

Клієнтську частину розроблено на основі JavaScript фреймворку Angular 2+, який дозволяє створювати веб-сторінки типу SPA (Single Page Application), тобто ті, що працюють з динамічним контентом без перезавантаження сторінки браузера.

Сервер було реалізовано за допомогою Node.js. Вхідні дані передаються з клієнта у вигляді JSON об’єкту засобами REST API.

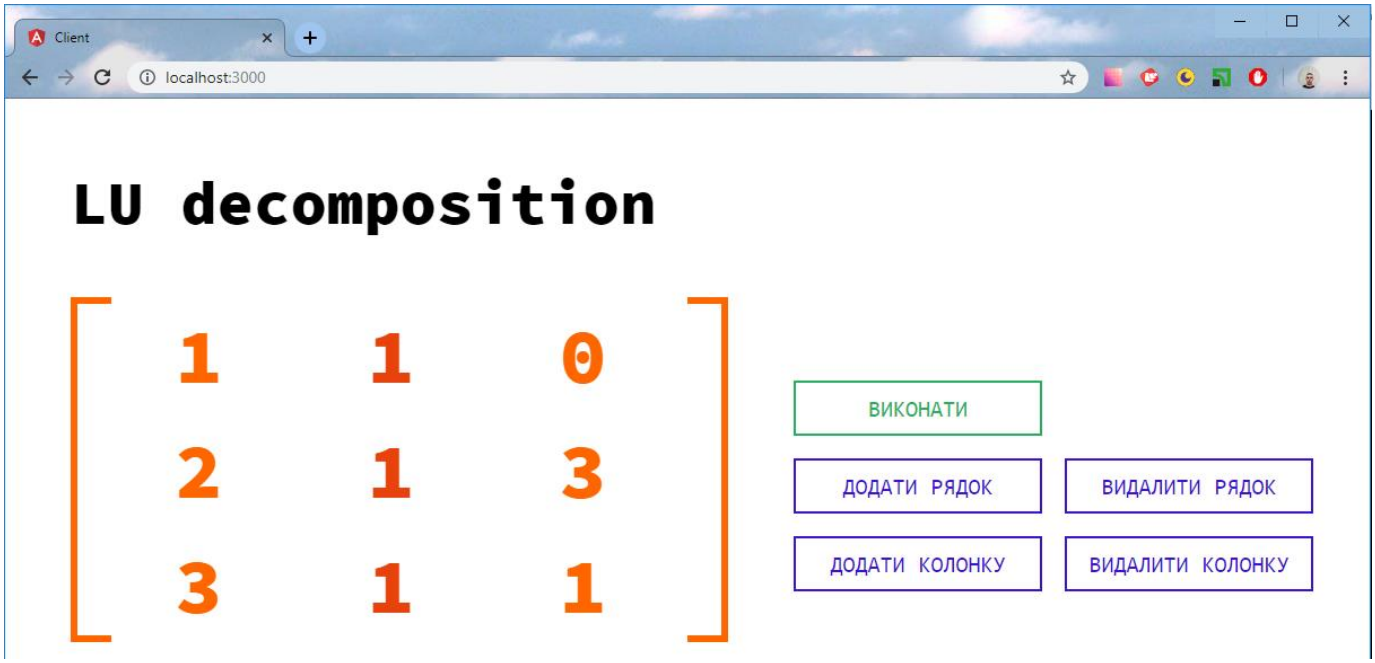

Рис. 4.1. Вигляд клієнту

Також користувач може редагувати значення в матриці (за допомогою стрілок, які розташовані зправа у блоці з значенням, або ж вводити вручну), додавати чи видаляти нові рядки та колонки (Рис. 4.2). Цю можливість реалізовано за допомогою Angular Forms [35].

Форми поділяються на два види – реактивні та динамічні. В сервісі використовувались реактивні форми, так як з ними легше та зрозуміліше працювати. Переваги використання реактивних форм в тому, що коли користувач змінює число

в матриці, воно автоматично змінюється в кінцевому об’єкті, який буде надсилатись на сервер.

Коли користувач нажимає на кнопку, викликається функція–обробник, яка опрацьовує об’єкт з форми, та надсилає його на сервер.

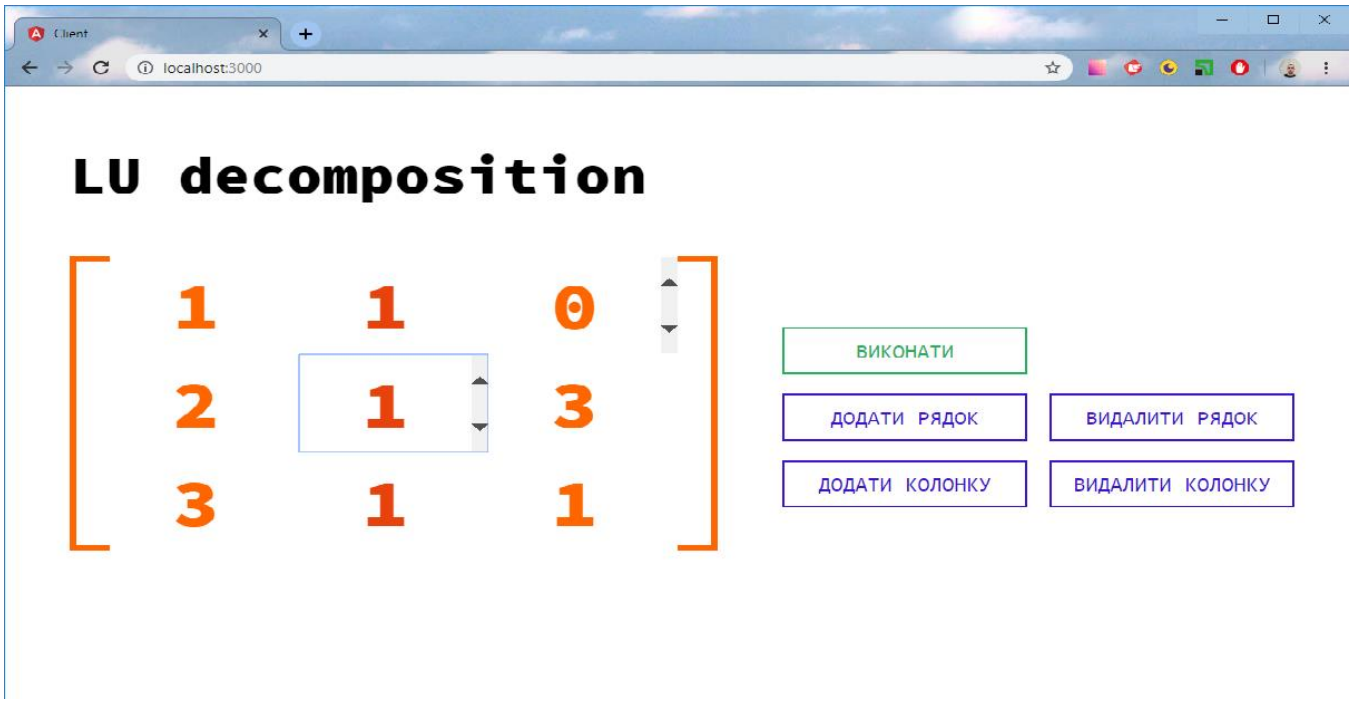

Рис. 4.2. Введення матриці

Після того, як сервер отримує запит клієнта, запускається функція з скомпільованої C++ динамічної бібліотеки. Виконуються обрахунки та вона повертає результат у вигляді верхньої та нижньої матриці. Також вимірюється час, за який ця функція відпрацювала. Після цього цей самий алгоритм, тільки написаний за допомогою Javascript мови програмування запускається, результат повертається у вигляді двох матриць, також вимірюється час, за який функція відпрацювала. Після виконання двох функцій, вимірювання часу, формується результуючий об’єкт, в якому вказані результати скомпільованої функції та нативної та час виконання кожної. Результуючий об’єкт надсилається для клієнта та в подальшому опрацьовується вже на стороні клієнта.

В об'єкті є поле “results”, в якому є два поля: “cpp” та “js”, у які відповідно вкладено час виконання коду кожної мови програмування та результат. Результат вкладено для того, щоб впевнетись, що результати двох функцій однакові.

На рис. 4.3 зправа знизу виведено час виконання функцій. Час функції з скомпільованої бібліотеки C++ складає 0,0383 мілісекунди, в той час як Javascript функція відпрацювала за 0,4489 мілісекунди. Звідси можна зробити висновок, що функція C++ повернула результат у 11.7 разів швидше ніж та сама функція, але написана за допомогою нативного Javascript.

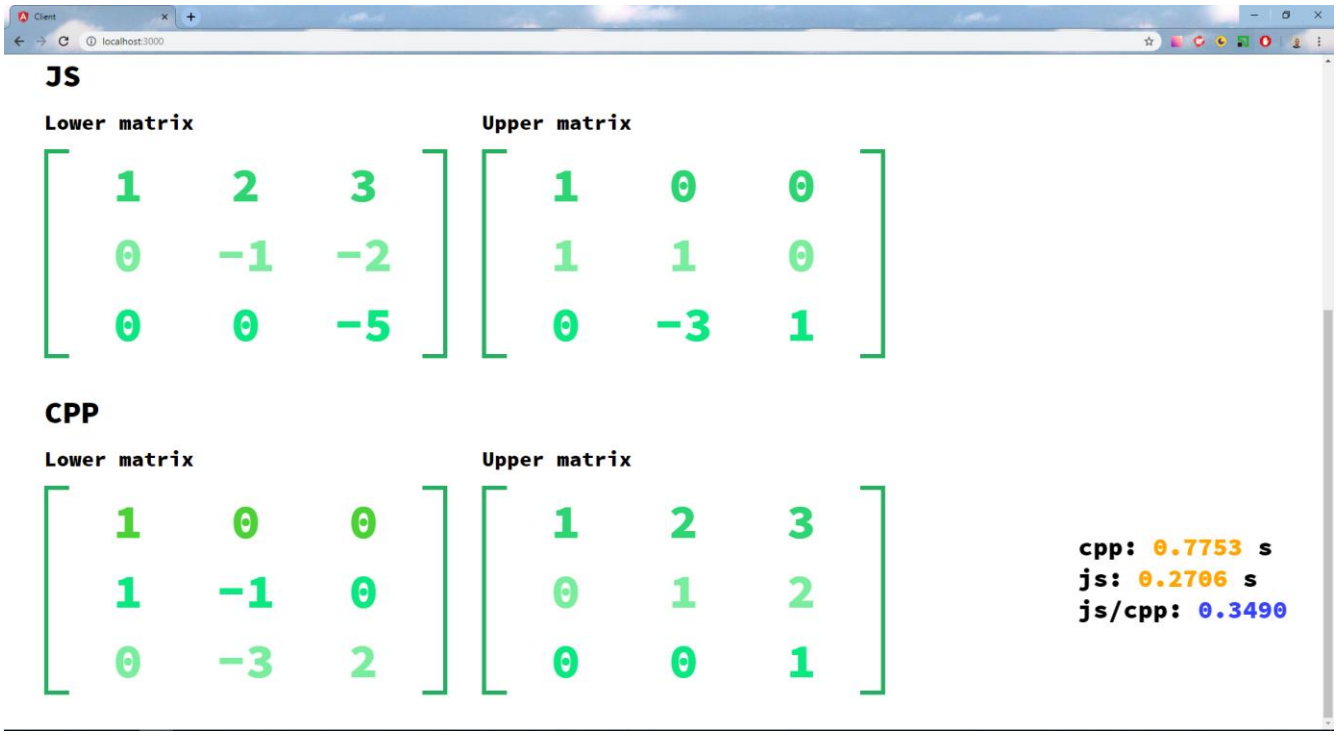

Рис. 4.3. Результат виконання запиту

Також проводились заміри виконання алгоритму на 100, 1000 та 10000 разів. В середньому результати були такі самі, але виконання C++ функції відносно Javascript, відбувалось переважно у 1-2 разів швидше.

Також варто взяти до уваги технічні характеристики машини, на якій виконується код (знаходиться сервер) та процеси, які могли працювати на фоні, тим самим, примушували функції відпрацьовувати довше. Зазвичай для серверу виділяють окрему машину, на якій працює лише сервер та супутні програми/засоби,

які необхідні для їх роботи. Тому очікується приріст у швидкодії ще у декілька разів, коли коли сервер буде розташовано на окремій машині (хостингу).

Виконано сервіс, який можна використовувати для компіляції раніше написаних динамічних бібліотек для подальшого використання у Node.js сервісі. Також можна створювати нові бібліотеки за допомогою C++ мови програмування для збільшення продуктивності коду, використання переваг, багатопоточності, строгої типізації.

Сервісом можуть користуватись розробники та компілювати бібліотеки. Також скомпільованою бібліотекою можна поділитись з іншими розробниками, щоб не встановлювати необхідні утиліти для компіляції на всі машини.

У подальшому планується удосконалення клієнту та серверу. Створення повноціного веб-сервісу хмарних обчислень. Саме для цієї цілі написано C++ код, який більш продуктивно буде виконувати багато громіздких обчислень, по запиту клієнта, та компілюватись у бібліотеку.

Також для клієнту та серверу буде удосконалено код, для того, щоб одночасно могли виконувати один і той самий запит декілька користувачів. Поділ серверу на мікросервери, для кращої продуктивності.

На разі, розробники можуть використувати цей веб-сервіс, як початкову точку у своїх проектах, де вже все налаштовано та підготовлено для використання та компіляції C++ коду у бібліотеку.

Також можна зробити висновок, що компіляція C++ коду у бібліотеку дає кращий результат ніж написання власного модулю для Node.js. Адже для цього необхідні знання Nan (native addons for Node.js), що є додатковою бібліотекою для C++, та має достатній поріг входу. А написання бібліотеки економить значно час та людські ресурси.

ВИСНОВКИ

У результаті виконання роботи зроблено наступні висновки:

1. Проаналізовано існуючі шляхи інтеграції C++ коду у середовище серверу Node.js, та обрано інтеграцію у вигляді динамічної бібліотеки DLL, яка компілюється з використанням пакету Node-gyp.
2. Розроблено архітектуру взаємодії клієнта з сервером, яка надала можливість проводити математичні розрахунки на сервері без встановлення програмного забезпечення на клієнті.
3. Реалізовано веб-сервіс, в який інтегровано бібліотеку з функцією розрахунку LU декомпозиції матриці. Показано, що швидкість виконання функції на C++ коді є подібною до функції на мові Javascript.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. TIOBE C. TIOBE Index for March 2019 [Електронний ресурс] / COMPANY TIOBE // TIOBE. – 2019. – Режим доступу до ресурсу: https://www.tiobe.com/tiobe-index/.
2. C++ Documentation [Електронний ресурс] // 06.05.2019. – 2019. – Режим доступу до ресурсу: https://docs.microsoft.com/ru-ru/cpp/cpp/cpp-language-reference?view=vs-2019.
3. Node.js Documentation [Електронний ресурс]. – 2009. – Режим доступу до ресурсу: https://nodejs.org/uk/.
4. CGI for beginners [Електронний ресурс]. – 2018. – Режим доступу до ресурсу: https://lectureswww.readthedocs.io/5.web.server/cgi.html.
5. CGI history [Електронний ресурс]. – 1976. – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/CGI.
6. CGI [Електронний ресурс] // 2019 – Режим доступу до ресурсу: http://www.firststeps.ru/cgi/r.php?1.
7. Getting your C++ to the Web with Node.js [Електронний ресурс]. – 2015. – Режим доступу до ресурсу: https://nodeaddons.com/getting-your-c-to-the-web-with-node-js/?fbclid=IwAR0jXFPe9QvzLeCUU14LHaLwt_hHZizzckQheB0XkpF_xP4RovFP4gKNuuY.
8. Automating a C++ program from a Node.js web app [Електронний ресурс]. – 2015. – Режим доступу до ресурсу: https://nodeaddons.com/automating-a-c-program-from-a-node-js-web-app/.
9. Calling Native C++ DLLs from a Node.js Web App [Електронний ресурс]. – 2015. – Режим доступу до ресурсу: https://nodeaddons.com/calling-native-c-dlls-from-a-node-js-web-app/.

10. Building an Asynchronous C++ Addon for Node.js using Nan [Електронний ресурс]. – 2015. – Режим доступу до ресурсу: https://nodeaddons.com/building-an-asynchronous-c-addon-for-node-js-using-nan/.

11. Native Abstractions for Node.js [Електронний ресурс]. – 2015. – Режим доступу до ресурсу: https://github.com/nodejs/nan.

12. JS Modules [Електронний ресурс] // 2017 – Режим доступу до ресурсу: https://learn.javascript.ru/modules.

13. What is V8? [Електронний ресурс] // 2018 – Режим доступу до ресурсу: https://v8.dev/.

14. The Chromium Projects [Електронний ресурс] // 2014 – Режим доступу до ресурсу: https://www.chromium.org/.

15. Объектно-ориентированное программирование [Електронний ресурс] // 2015 – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/ООП

16. Libuv overwiew [Електронний ресурс] // 2009 – Режим доступу до ресурсу: https://github.com/libuv/libuv.

17. POSIX [Електронний ресурс] // 2004 – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/POSIX.

18. Emscripten documentation [Електронний ресурс] // 2001 – Режим доступу до ресурсу: https://github.com/emscripten-core/emscripten.

19. LLVM Overview [Електронний ресурс] // 2015 – Режим доступу до ресурсу: https://llvm.org/.

20. LLVM Design & Overview [Електронний ресурс] // 2016 – Режим доступу до ресурсу: https://llvm.org/docs/.

21. Welcome to SWIG [Електронний ресурс] // 2012 – Режим доступу до ресурсу: http://www.swig.org/.

22. SWIG [Електронний ресурс] // 2003 – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/SWIG.

23. Node.js native addon build tool [Електронний ресурс] // 2010 – Режим доступу до ресурсу: https://github.com/nodejs/node-gyp.

24. GNU Make [Електронний ресурс] // 1993 – Режим доступу до ресурсу: https://www.gnu.org/software/make/.

25. Compiling with g++ [Електронний ресурс] // 2001 – Режим доступу до ресурсу: https://www.geeksforgeeks.org/compiling-with-g-plus-plus/.

26. Python documentation [Електронний ресурс] // 1995 – Режим доступу до ресурсу: https://www.python.org/.

27. Angular Documentation [Електронний ресурс] // 2012 – Режим доступу до ресурсу: https://angular.io/.

28. TypeScript Tutorial [Електронний ресурс] // 2009 – Режим доступу до ресурсу: https://www.tutorialspoint.com/typescript/.

29. Electronjs documentation [Електронний ресурс] // 2014 – Режим доступу до ресурсу: https://electronjs.org/.

30. L U Decomposition of a System of Linear Equations [Електронний ресурс] // 2011 – Режим доступу до ресурсу: https://www.geeksforgeeks.org/l-u-decomposition-system-linear-equations/.

31. Generate Your Projects. [Електронний ресурс] // 1980 – Режим доступу до ресурсу: https://gyp.gsrc.io/.

32. Node FFI Tutorial [Електронний ресурс] // 2016 – Режим доступу до ресурсу: https://github.com/node-ffi/node-ffi/wiki/Node-FFI-Tutorial.

33. ref [Електронний ресурс] // 2016 – Режим доступу до ресурсу: https://www.npmjs.com/package/ref.

34. REST API [Електронний ресурс] // 2000 – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/REST.

35. Forms overview [Електронний ресурс] // 2012 – Режим доступу до ресурсу: https://angular.io/guide/forms-overview.

36. Baraniecki M. Extending Node.js with native C++ modules [Електронний ресурс] / Marcin Baraniecki // Medium. – 2017. – Режим доступу до ресурсу: https://medium.com/@marcinbaraniecki/extending-node-js-with-native-c-modules-63294a91ce4.

Додаток А

Інтеграція динамічної бібліотеки математичних розрахунків з веб-сервісом Node.js

Специфікація

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТР-5286_19Б

Аркушів 1

Київ 2019

Позначення	Найменування	Примітки
Документація		
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТР5286_19Б	Записка.docx	Текстова частина дипломної роботи
Компоненти		
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТР5286_19Б 1	server	Основна папка, яка містить необхідні файли для запуску сервера
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТР5286_19Б	client	Основна папка, яка містить файли для клієнта

Додаток Б

Інтеграція динамічної бібліотеки математичних розрахунків з веб-сервісом Node.js

Текст програми

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТР-5286_19Б

Аркушів 2

Київ 2019

Текст програми взаємодії Node.js з бібліотекою

```
const express = require("express");
const app = express();
var ffi = require('ffi')
var ref = require('ref')
var ArrayType = require('ref-array')
const cors = require('cors');
var cluster = require('cluster');

const performance = require('perf_hooks').performance;
const LU = require('./lu');
if (cluster.isMaster) {
    var cpuCount = require('os').cpus().length;
    for (var i = 0; i < cpuCount; i += 1) {
        cluster.fork();
    }
} else {
    const port = process.env.PORT || 3000;
    app.use(express.static(__dirname + '/dist/client'));
    app.get('/*', function (req, res) {
        res.sendFile(path.join(__dirname + '/dist/client/index.html'));
    });
    app.use(cors());
    app.use(express.json());
    app.post("/calculate", (req, res) => {
        const { array } = req.body;
        var float = ref.types.float;
        var floatArrayType = ArrayType(float);
        var floatArr = new floatArrayType(array.length);
        var int = ref.types.int;
        var library = ffi.Library('./build/Release/result', {
            "LUdecomposition": [int, [floatArrayType, floatArrayType, floatArrayType, int, int]]
        })
        const readyToUseArray = new floatArrayType(prepareArray(array));
        let l = new floatArrayType(readyToUseArray.length);
        let u = new floatArrayType(readyToUseArray.length);
        var t1 = performance.now();
```

```
        library.LUdecomposition(readyToUseArray, l, u, array.length, array[0].length);
        const cppTime = performance.now() - t1;
        t1 = performance.now();
        const jsResult = LU(array);
        const jsTime = performance.now() - t1;
        res.send({ code: 200, results: { js: { time: jsTime, result: jsResult }, cpp: { time: cppTime, result: { lower: l, upper: u } } } });
    });
    app.listen(port, (req, res) => {
        console.log(`server has started at ${port} port`)
    });
}
function prepareArray(array) {
    const resArray = [];
    for (let i = 0; i < array.length; i++) {
        for (let j = 0; j < array[0].length; j++) {
            resArray.push(array[i][j]);
        }
    }
    return resArray;
}
```

Додаток В

Інтеграція динамічної бібліотеки математичних розрахунків з веб-сервісом Node.js

Опис програми

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТР-5286_19Б

Аркушів 8

Київ 2019

АНОТАЦІЯ

Даний додаток містить опис веб-сервісу розробленого для інтеграції динамічної бібліотеки математичних розрахунків. Створено веб-сервіс типу “ клієнт – сервер”. Сервіс виконує наступні завдання:

1. Введення даних для обчислення
2. Надсилання даних на сервер
3. Обчислення даних на сервері
4. Надсилання результату обрахунків на клієнт.

При розробці веб-сервісу використовувався мова програмування Javascript, середовище Node.js та фреймворк Angular 2+ (в основі яких лежить Javascript).

ЗМІСТ

ЗАГАЛЬНІ ВІДОМОСТІ

У цьому додатку міститься опис веб-сервісу у який інтегровано бібліотеку динамічних розрахунків з веб-сервісом Node.js. У додатку Б міститься програмний код головних модулів розроблюваного серверу.

Веб-сервіс виконано для подальшої інтеграції на серверну віддалену машину. Для коректної роботи серверу необхідно встановити всі залежні модулі на сервер, в тому числі і Node-gyp.

При розробці веб-сервісу використовувалась мова Javascript з використанням середовища Microsoft Visual Studio Code.

ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Розроблений сервіс виконує завдання інтеграції динамічної бібліотеки написаній за допомогою C++ мови у середовище Node.js та користування функціями з бібліотеки.

Розроблений сервіс може використовуватись в якості учбових матеріалів при опрацюванні технологій C++, Javascript мов програмування. Функціональних обмеження на використання веб-сервісу полягає лише в діапазоні введення даних для обчислення швидкості.

ОПИС ЛОГІЧНОЇ СТРУКТУРИ

Для реалізації взаємодії C++ коду з Node.js середовищем необхідно спочатку визначитись яким методом необхідно інтегрувати C++ коду. У роботі було вибрано за допомогою бібліотеки. Згодом код C++ необхідно скомпілювати засобами Node-gyp. Отриману бібліотеку завантажуємо у Node.js середовище засобами FFI.

При запиті клієнта до сервера, запускається функція обробник, яка в свою чергу, запускає функцію з скомпільованої бібліотеки та надсилає результат для клієнта.

ВИКОРИСТОВУВАНІ ТЕХНІЧНІ ЗАСОБИ

Для забезпечення повноцінної роботи та досягнення високої ефективності роботи веб-сервісу для демонстрації взаємодії C++ і Node.js у процесі інтеграції динамічної бібліотеки було обрано Visual Studio Code яка показала себе надійним та гнучким середовищем розробки програм.

Розроблений веб-сервіс працює на будь-якій платформі у будь-якому браузері. У подальшому необхідно захостити веб-сервіс на віддаленому сервері для його стабільної роботи та доступу у мережі інтернет.

ВИКЛИК І ЗАВАНТАЖЕННЯ

Розроблений веб-сервіс потребує інсталяції. Для того, щоб запустити сервер необхідно спочатку встановити всі залежності командою “npm i” в консолі. Після чого необхідно запустити сервер командою “node app”.

Для інсталяції клієнту необхідно також встановити залежності тією самою командою, що і для серверу. Після чого необхідно написати команду “ng s -o” та клієнт буде завантажено автоматично у браузері.

ВХІДНІ І ВИХІДНІ ДАНІ

Вхідними даними є квадратна матриця, яка передається у фунції обчислення LU декомпозиції.

Вихідними даними є масив двох матриць: верхня та нижня. Вони є результатом функції обчислення LU декомпозиції.

Додаток Г

Інтеграція динамічної бібліотеки математичних розрахунків з веб-сервісом Node.js

Апробації

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТР-5286_19Б

Аркушів 3

Київ 2019

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

СУЧАСНІ ПРОБЛЕМИ НАУКОВОГО ЗАБЕЗПЕЧЕННЯ ЕНЕРГЕТИКИ

Матеріали XVII Міжнародної науково-практичної конференції молодих вчених та студентів м. Київ, 23-26 квітня 2019 року,

ТОМ 2

Київ- 2019

УДК 004.45

Студент 4 курсу, гр. ТР-52 Піддубняк А.В.
Доц., к.т.н. Демчишин А.А.

ІНТЕГРАЦІЯ ДИНАМІЧНОЇ БІБЛІОТЕКИ МАТЕМАТИЧНИХ РОЗРАХУНКІВ З ВЕБ-СЕРВІСОМ NODE.JS

На даний час технічного та інформаційного розвитку існує більше двадцяти популярних мов програмування [1] та технологій, що полегшують роботу з ними. Деякі з цих мов мають доволі схожу семантику, в той же час їх суттєва різниця полягає в можливостях, і, як наслідок, сферах застосування: наприклад, одні мови спеціалізуються для веб-розробки (JavaScript), інші (C#, C++) для створення десктоп додатків. У зв'язку з цим виникає потреба у раціональному використанні можливостей/переваг/швидкодії кількох мов в контексті одного проекту.

Метою роботи є створення веб-сервісу за допомогою мови програмування JavaScript, який на запит клієнта виконує методи бібліотеки математичних розрахунків, що реалізовано на мові C++, та повертає результат користувачеві.

В контексті даної роботи інструментом реалізації архітектури "клієнт–сервер" обрано програмну платформу Node.js [2], яка дає змогу ефективно використовувати REST API (Representational State Transfer) парадигму. В якості прикладу, підготовлено математичну бібліотеку, що виконує LU декомпозицію. Код математичних методів виконується на сервері у скомпільованому вигляді DLL бібліотеки, що є прив'язаною до сервісу. Для компіляції використано інструмент node-gyp (альтернативні інструменти: Emscripten, swig) [3].

Клієнтську частину розроблено на основі JavaScript фреймворку Angular 2+, який дозволяє створювати веб-сторінки типу SPA (Single Page Application), тобто ті, що працюють з динамічним контентом без перезавантаження сторінки браузера.

Схема роботи програмної системи виглядає наступним чином:

1. Клієнт робить запит на сервер використовуючи клієнтську сторінку (запит надсилається у відповідності до REST API).
2. Сервер, що реалізовано на мові JavaScript, приймає запит клієнта.
3. Серверна частина опрацьовує отримані дані, та запускає скомпільовану функцію на мові C++.
4. Сервер надсилає отриманий результат клієнту засобами REST API у вигляді нижньої та верхньої трикутної матриці.

В роботі показано можливість використання існуючого коду для реалізації ресурсоємних розрахунків (LU декомпозиція потребує $2n^3 / 3$ арифметичних операцій) у вигляді веб-сервісу із сучасним прозорим інтерфейсом.

Перелік посилань:

1) TIOBE C. TIOBE Index for March 2019 [Електронний ресурс] / COMPANY TIOBE // TIOBE. – 2019. – Режим доступу до ресурсу: https://www.tiobe.com/tiobe-index/.

2) Hahn E. Express in Action / Evan Hahn. – Shelter Island, NY 11964: Manning Publications Co., 2016. – 217 p.

3) Baraniecki M. Extending Node.js with native C++ modules [Електронний ресурс] / Marcin Baraniecki // Medium. – 2017. – Режим доступу до ресурсу: https://medium.com/@marcinbaraniecki/extending-node-js-with-native-c-modules-63294a91ce4.

Додаток Д

Інтеграція динамічної бібліотеки математичних розрахунків з веб-сервісом Node.js

Нагороди

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТР-5286_19Б

Аркушів 1

Київ 2019

ВІДГУК

керівника дипломної роботи

освітньо-кваліфікаційного рівня „бакалавр”

виконаної на тему : Інтеграція динамічної бібліотеки математичних розрахунків з веб-сервісом Node.js

студентом (кою) __Піддубняком А.В.

(прізвище, ім’я, по батькові)

(складається у довільній формі із зазначенням: головної цілі дипломної роботи, в інтересах або на замовлення якої організації він виконаний (в рамках науково дослідної роботи кафедри, підприємства, НДІ тощо); відповідності виконаної ДР завданню; ступеня самостійності при виконанні ДР; рівня підготовленості дипломника до прийняття сучасних рішень; уміння аналізувати необхідні літературні джерела, приймати правильні(інженерні, наукові) рішення, застосовувати сучасні системні та інформаційні технології, проводити фізичне або математичне моделювання, обробляти та аналізувати результати експерименту; найбільш важливих теоретичних та практичних результатів апробації їх(участь у конференціях, семінарах, оформлення патентів, публікація в наукових журналах тощо); загальної оцінки виконаної ДР, відповідності якості підготовки дипломника вимогам ОКХ і можливості присвоєння йому відповідної кваліфікації; інші питання, які характеризують професійні якості дипломника)

Керівник дипломної роботи

_____	_____	_____
(посада, вчені звання, ступінь)	(підпис)	(ініціали, прізвище)

РЕЦЕНЗІЯ

на дипломну роботу

освітньо-кваліфікаційного рівня „бакалавр”

виконаної на тему : Інтеграція динамічної бібліотеки математичних розрахунків з веб-сервісом Node.js

студентом Піддубняком А.В.

(прізвище, ім’я, по батькові)

При розробці виникають ситуації, коли необхідно використовувати переваги різних мов у контексті одного проекту. Ці дії потребують досить серйозних затрат часу, грамотне керування цими діями може суттєво вплинути на час роботи, а отже і продуктивність системи. Тому вирішення задачі, поставленої в рамках дипломної роботи є досить актуальною проблемою.

У поданій на рецензію дипломній роботі розглянуті основні засоби створення клієнт-серверного додатку, проаналізовано їх переваги та недоліки, визначено особливості та переваги платформи Node.js.

Наведений опис програмного продукту вказує на те, що розроблений в рамках дипломної роботи модуль математичних розрахунків дозволяє зменшити навантаження на сервер та використовувати переваги C++ мови програмування.

Робота виконана на високому рівні та повністю відповідає вимогам, які були поставлені до дипломного проекту.

Вцілому робота виконана на високому технічному рівні, а студент Піддубняк А.В. заслуговує присвоєння кваліфікації «Інженера з компютерних систем» за спеціальністю 7.050101 «Комп’ютерні науки».

Рецензент

_____	_____	_____
(посада, вчені звання, ступінь)	(підпис)	(ініціали, прізвище)